

# 1 基于顺序存储结构的线性表实现

## 问题描述

为实现基于顺序存储结构的线性表及以此为基础的多表管理和文件读写功能，按如下形式设计数据的逻辑结构及基本运算。

### 线性表抽象数据类型

依据最小完备性和常用性相结合的原则，设计了线性表的数据对象和数据关系，并定义了线性表的初始化表、销毁表、清空表、判定空表、求表长和获得元素等12种基本运算，具体数据和运算功能定义如下。

**ADT** SqList {

**数据对象：**

**数据关系：**

**基本操作：**

SqList\_init(&L)

初始条件：线性表L不存在。

操作结果：构造一个空的线性表。

SqList\_destroy(&L)

初始条件：线性表L已存在。

操作结果：销毁线性表L。

SqList\_clear(&L)

初始条件：线性表L已存在。

操作结果：将L重制为 空表。

SqList\_empty(L)

初始条件：线性表L已存在。

操作结果：若L为空表则返回TRUE，否则返回FALSE。

SqList\_length(L)

初始条件：线性表L已存在。

操作结果：返回L中数据元素的个数。

SqList\_getElem(L, idx, &ele)

初始条件：线性表L已存在且非空， 。

操作结果：用ele返回L中第idx个数据元素的值。

SqList\_locateElem(L, val)

初始条件：线性表L已存在且非空。

操作结果：返回L中第一个值为val的数据元素的位序。若这样的元素不存在则返回0。

SqList\_priorElem(L, cur, &pre)

初始条件：线性表L已存在、非空且cur不是第一个数据元素。

操作结果：若cur是L的数据元素，则将它的前驱存入pre，否则返回FALSE。

SqList\_nextElem(L, cur, &next)

初始条件：线性表L已存在、非空且cur不是最后一个数据元素。

操作结果：若cur是L的数据元素，则将它的后继存入next，否则返回FALSE。

SqList\_insert(&L, key, val)

初始条件：线性表L已存在且非空， 。

操作结果：在L的第key个位置之前插入新的数据元素val。

SqList\_delete(&L, key, &val)

初始条件：线性表L已存在且非空， 。

操作结果：删除L的第key个数据元素，其值由val带回。

SqList\_traverse(L)

初始条件：线性表L已存在。

操作结果：依次访问并输出L的每个数据元素。

} **ADT** SqList

### 1.1.2 多线性表操作

另外了设计了多线性表操作，可以在多个互相独立的线性表中切换工作表。

### 1.1.3 演示系统与数据文件

直接以实验文档中提供的框架作为演示系统。该框架将完成函数调用所需实参值的准备和函数执行结果的现实，并给出适当的操作提示显示。

演示系统可实现线性表的文件形式保存。其中，需要设计文件数据记录格式，以高效保存线性表数据逻辑结构的完整信息；需要设计线性表文件保存和加载操作合理模式。

演示系统还支持选择多个线性表进行管理。

## 1.2 系统设计

### 1.2.1 数据物理结构

1. 线性表的数据存储结构

线性表数据类型在程序中定义为结构体SqList ，具体定义方式如下：

typedef struct \_SqList {

ElemType \*elem; // 指向线性存储区

int length; // 表长

int list\_size; // 表容量

} SqList;

1. 多线性表的数据存储结构

多线性表操作通过定义数据元素类型为线性表SqList的数组，并设置一个整型常量用以保存工作表在线性表数组中的位置实现，具体定义方式如下：

SqList list\_pool[LIST\_POOL\_SIZE]; // 表池

int current\_list = 0; // 保存 L 在 list\_pool 中的位置

在本程序中，数据原子类型ElemType被定义为int整型。

### 1.2.2 演示系统

演示系统由用户操作界面与功能调用部分组成。

用户操作界面输出可选的线性表操作项、操作的结果及错误信息，基于c语言的标准输入输出库（stdio.h）及控制台的clear命令实现。

功能调用部分将用户输入的有关信息传递给线性数据结构的操作函数进行调用，并对函数的返回值进行处理判断输出相应的提示信息。

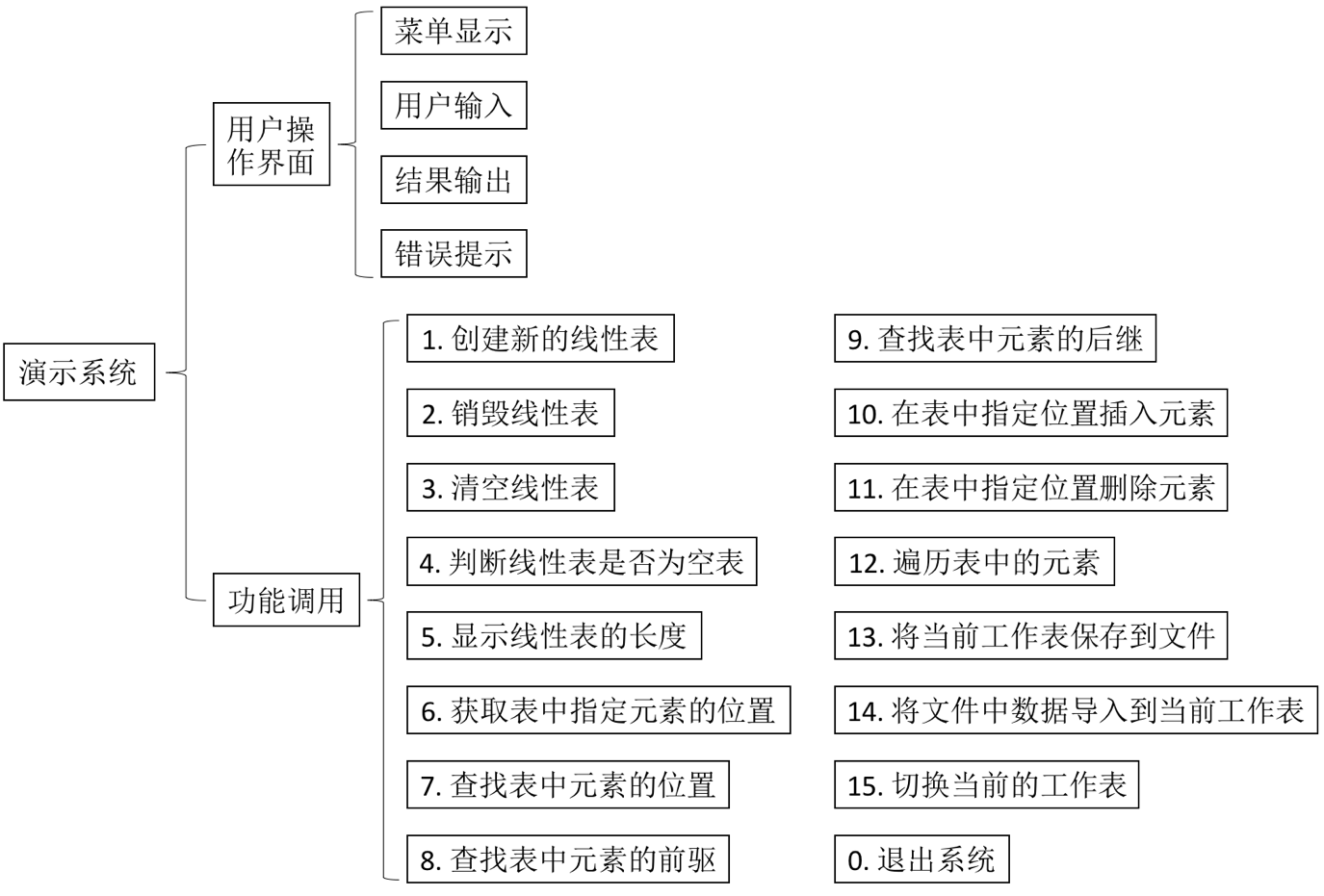


图1.2.2-1 演示系统模块结构图

### 1.2.3 数据文件存储格式

考虑到简洁高效与直观实用的原则，按如下方式设计数据文件存储格式。

文件格式：二进制文件，可用hexdump等工具查看数据值。

存放方式：顺序存放数据值的机器码，即直接将内存区块写入文件中。

### 1.2.4 线性表运算实现算法

1. SqList\_init(&L)

**算法思想：**首先判断L->elem是否为空指针，若不是则返回ERROR；否则使用malloc函数分配LIST\_INIT\_SIZE个ElemType类型的存储空间并使L->elem指向这一段空间，若分配失败则返回ERROR，否则L->length = 0，L->list\_size = LIST\_INIT\_SIZE，返回OK。

**时空效率分析：**算法的时间复杂度为常量阶O(1) 。

1. SqList\_destroy(&L)

**算法思想：**首先判断L->elem是否为空指针，若是则返回ERROR；否则使用free函数释放L->elem指向的存储空间，并L->elem置空，L->length = 0，L->list\_size = 0。

**时空效率分析：**算法的时间复杂度为常量阶O(1) 。

1. SqList\_clear(&L)

**算法思想：**首先判断L->elem是否为空指针，若是则返回ERROR；否则直接L->length = 0，再判断L->list\_size是否大于LIST\_INIT\_SIZE，若是则给L->elem重新分配大小为LIST\_INIT\_SIZE的空间，若分配失败则返回ERROR，否则L->list\_size = LIST\_INIT\_SIZE，返回OK。

**时空效率分析：**算法的时间复杂度为常量阶O(1) 。

1. SqList\_empty(L)

**算法思想：**首先判断L->elem是否为空指针，若是则返回ERROR；然后判断L.length是否为零，若是则返回TRUE，否则返回FALSE。

**时空效率分析：**算法的时间复杂度为常量阶O(1) 。

1. SqList\_length(L)

**算法思想：**首先判断L->elem是否为空指针，若是则返回ERROR；否则返回L.length。

**时空效率分析：**算法的时空复杂度为常量阶O(1) 。

1. SqList\_getElem(L, key, &val)

**算法思想：**首先判断L.elem是否为空指针，若是则返回ERROR；否则继续判断key是否在区间 [1, L.length] 之间，若不是则返回ERROR，否则将L.elem[key – 1] 赋值给val，返回OK。

**时空效率分析：**算法的时间复杂度为常量阶O(1) 。

1. SqList\_locateElem(L, val)

**算法思想：**首先判断L.elem是否为空指针，若是则返回ERROR；否则遍历L.elem中的每一个元素，直到找到第一个值为val的元素，返回该元素的位序。若遍历后未找到值为val的元素，则返回0。

**时空效率分析：**算法中有一层循环，由于第一个值为val的元素可能在L.elem中的任意位置出现，也可能不出现，故算法的时间复杂度为线性阶O(n) 。

1. SqList\_priorElem(L, cur\_e, &pre\_e)

**算法思想：**首先判断L.elem是否为空指针，若是则返回ERROR；否则调用SqList\_locate(L, cur\_e) 获取L.elem中第一个值为cur\_e的元素的位序，若返回值为0（即表中没有值为cur\_e的元素）或1（表中第一个元素没有前驱）则返回ERROR，否则将该元素前驱的值赋给pre\_e带回，返回OK。

**时空效率分析：**算法调用了时间复杂度为线性阶O(n) 的SqList\_locateElem(…) ，除此之外再没有循环，故该算法的时间复杂度也为线性阶O(n) 。

1. SqList\_nextElem(L, cur\_e, &next\_e)

**算法思想：**首先判断L.elem是否为空指针，若是则返回ERROR；否则调用SqList\_locate(L, cur\_e) 获取L.elem中第一个值为cur\_e的元素的位序，若返回值为0（即表中没有值为cur\_e的元素）或L.length（表中最后一个元素没有后继）则返回ERROR，否则将该元素后继的值赋给next\_e带回，返回OK。

**时空效率分析：**类似于SqList\_priorElem(…) ，该算法的时间复杂度也为线性阶O(n) 。

1. SqList\_insert(&L, key, val)

**算法流程图：**

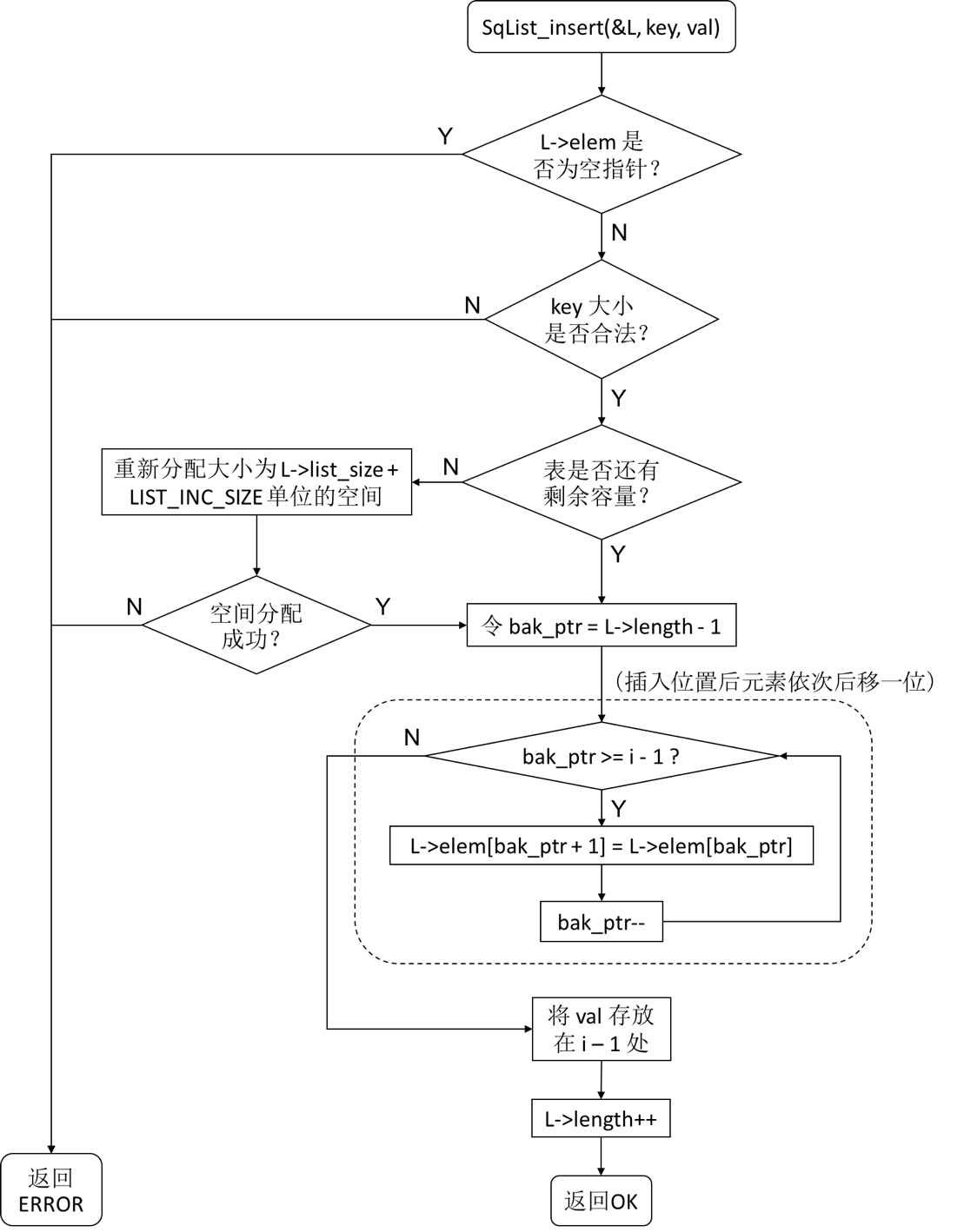


图1.2.4-1 SqList\_insert 算法流程图

**时空效率分析：**算法中要通过循环来移动元素，由于key值的大小不确定，故算法的时间复杂度为线性阶O(n) 。

1. SqList\_delete(&L, key, &val)

**算法思想：**与SqList\_insert(…) 算法类似，将由key指示位置的值由val带出后，将后面的元素依次前移。之后再检查L->list\_size - L->length是否大于LIST\_INC\_SIZE ，若是则重新分配大小为L->list\_size - LIST\_INC\_SIZE的空间，最后L->length-- ，返回OK。

**时空效率分析：**与SqList\_insert(…) 类似，算法的时间复杂度为线性阶O(n) 。

1. SqList\_traverse(L)

**算法思想：**首先判断L.elem是否为空指针，若是则返回ERROR；否则遍历L.elem并依次输出每一个数据元素，返回OK。

**时空效率分析：**算法的时间复杂度为线性阶O(n) 。

1. SqList\_saveList(fp, L)

**算法思想：**首先判断fp和L.elem是否为空指针，若是则返回ERROR；否则调用fwrite(…) 将内存区块写入fp指向的文件，返回OK。

**时空效率分析：**算法的时间复杂度为常量阶O(1) 。

1. SqList\_loadList(fp, &L)

**算法思想：**首先判断fp是否为空指针，若是则返回ERROR；再判断L->elem是否非空，若非空则返回ERROR（防止覆盖操作）。计算文件大小存入filesize中，为L->elem分配内存空间，调用fread(…) 直接读取filesize个数据单元的数据到L->elem中，返回OK。

**时空效率分析：**算法的时间复杂度为常量阶O(1) 。

### 1.2.5 多线性表运算实现算法

1. SqList\_selectList(&L, list\_pool, &cur\_l, tgt\_l)

**算法思想：**首先判断tgt\_l是否在合法的范围内，若不是则返回ERROR；否则将L下挂载的数据存到list\_pool[cur\_l] 中（cur\_l存放现操作的L在list\_pool中的位置），再将list\_pool[tgt\_l - 1] （tgt\_l为位序，下标从1开始）中的数据加载到L中，最后改变cur\_l的值为tgt\_l - 1 ，返回OK。

**时空效率分析：**算法的时间复杂度为常量阶O(1) 。

## 系统实现

### 1.3.1 实验环境

实验代码使用文本编辑器编写，clang（Apple LLVM version 9.0.0, clang-990.0.30）编译，源文件名为main.c 。

### 1.3.2 演示系统操作

演示系统采用键盘操作，控制台输出操作提示、可用的操作项与结果输出，输入不存在的操作编号不会执行任何操作。

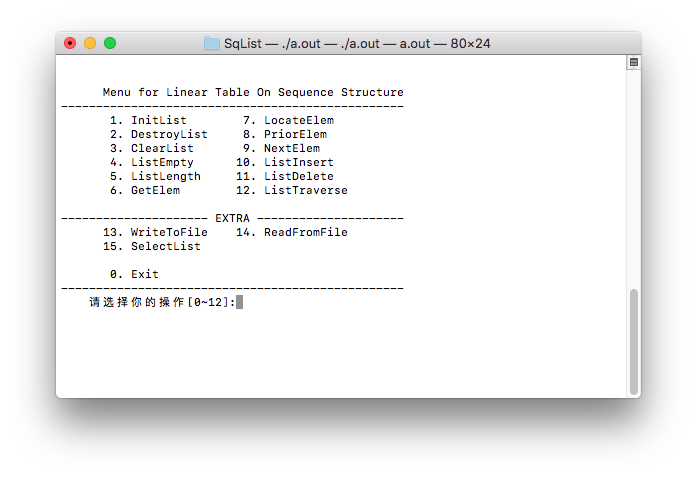


图1.3.2-1 启动程序后的操作菜单

输入操作指令0 ，程序退出。

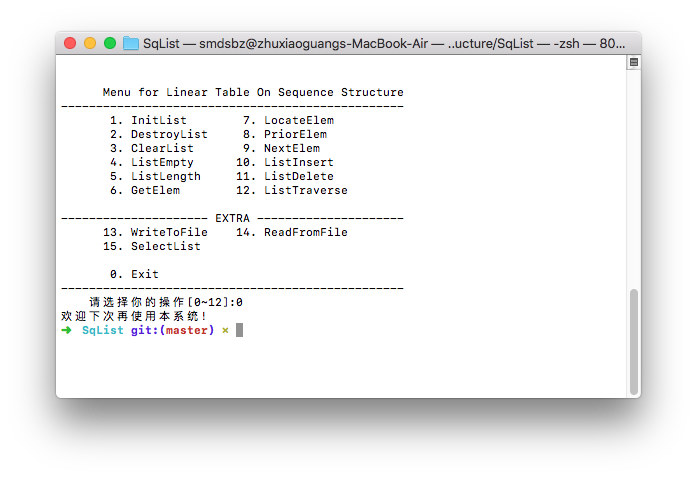


图1.3.2-2 输入操作0程序退出

### 1.3.3 测试计划

1. 对系统的单线性表操作功能按以下计划测试：
2. 检测常规状态下系统的工作情况；
3. 重点检查非法边界操作时的工作情况。

详见**表1.3.3-1**。

1. 对系统的多线性表管理功能按**表1.3.3-2**计划测试。